3 CONFIGURING A PROJECT TO USE THE VDK

In This Chapter

This chapter contains information about the VisualDSP++ Integrated Development and Debugging Environment (IDDE) support for the kernel. You can access the VDK components and services through the set of menus, commands, and windows in the development and debugging environment.

If you are new to VisualDSP++ DSP development software, we recommend to start with the *VisualDSP++2.0 Getting Started Guide* for your target processor family.

The IDDE support for the VDK can be broken into two areas:

- "Configuring a Project" on page 3-2
- "Debugging a VDK Project" on page 3-44

Configuring a Project

This section is designed so that you can quickly learn the VisualDSP++ environment as it applies to VDK-instrumented project development.

VisualDSP++ has been extended to manage all of the VDK components. You start developing a VDK-based application by creating a set of source files. The IDDE automatically generates source code framework for each user-requested VDK object. For example, when you request a new thread type by right-clicking the **Thread Type** icon on the **Kernel** tab in the **Project** window (see Figure 3-5 on page 3-8) and supplying a name for the new thread type, the environment generates a source file and a header file of that name. The automatically generated files contain all of the functions required by for a VDK Thread Type.

Linker Description File

When a new project makes use of the kernel, a reference to a VDK-specific default Linker Description File (`.LDF`) is added to the project. This file resides in an area of the VisualDSP++ install directory that is specific to your processor. Although the LDF may be used unchanged for many projects, you can copy the LDF from the VisualDSP++ install directory to your project directory, add it to the project, and modify it to suit your individual hardware configurations.

Thread Safe Libraries

Just as user threads must be reentrant, special "thread safe" versions of the standard C and C++ libraries are included for use with the VDK. The default `.LDF` included in VDK projects links with these libraries. If you modify your Linker Description File, ensure that the file links with the thread-safe libraries. Your project's LDF resides in the **Linker Files** folder and is accessible via the **Project** window, the **Project** tab of VisualDSP++ (see Figure 3-4 on page 3-7).

Header Files for the VDK API

When a VDK project is created in the development environment, one of the automatically generated files in the project directory is `vdk.h`. The header file contains enumerations for every user-defined object in the development environment and all VDK API declarations. Your source files must include `vdk.h` to access any kernel services.

VDK Program Development

All VDK program development within the IDDE includes the following steps:

- "Step 1: Start a New Project File" on page 3-4
- "Step 2: Modify VDK System Parameters" on page 3-8
- "Step 3: Add and Edit a Thread Type" on page 3-10
- "Step 4: Add and Edit a Boot Thread" on page 3-14
- "Step 5: Add and Edit a Round-Robin Priority" on page 3-18
- "Step 6: Add and Edit a Semaphore" on page 3-22
- "Step 7: Add and Edit an Event Bit" on page 3-25
- "Step 8: Add and Edit an Event" on page 3-28
- "Step 9: Add and Edit an Interrupt" on page 3-34
- "Step 10: Add and Edit a Device Driver" on page 3-38

By following these steps, your VDK projects build consistently and accurately with minimal project management. This process reduces development time and lets you to concentrate on algorithm development.

Step 1: Start a New Project File

To Create a VDK-Enabled Project

1. From the **Project** menu, choose **New...**

   VisualDSP++ opens the **Save New Project As** dialog box, prompting you to save the new project (Figure 3-1).

2. Fill in the project **File name:**, choose a **Save in:** directory, and click **OK** to save the project file, exiting the dialog box.

VDK support cannot be added to an existing project.

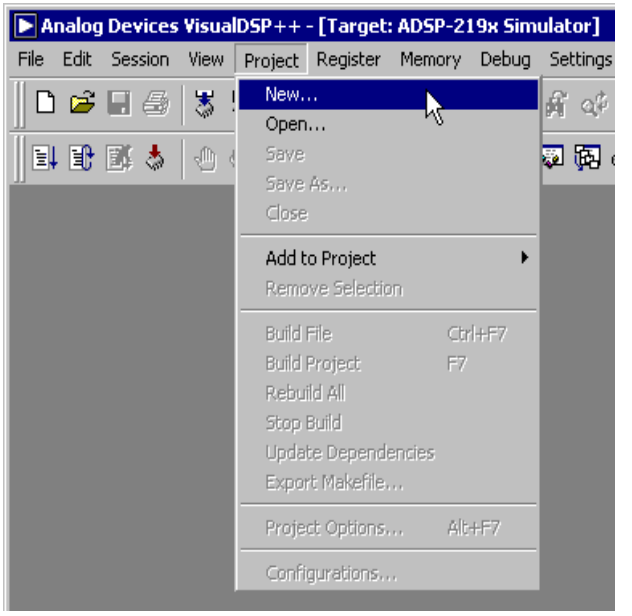

Figure 3-1. VisualDSP++ Environment: Project Menu

3. From the **Project** menu, choose **Project Options...**

   VisualDSP++ opens the **Project Options** dialog box, prompting you to set options for your project (Figure 3-2).

4. Set your target processor in the **Project Options** dialog box. This dialog sets the processor type for which VDK support has been installed. This example uses the `ADSP-2192-12` target processor. Click **OK** to apply the project option and exit the dialog box.

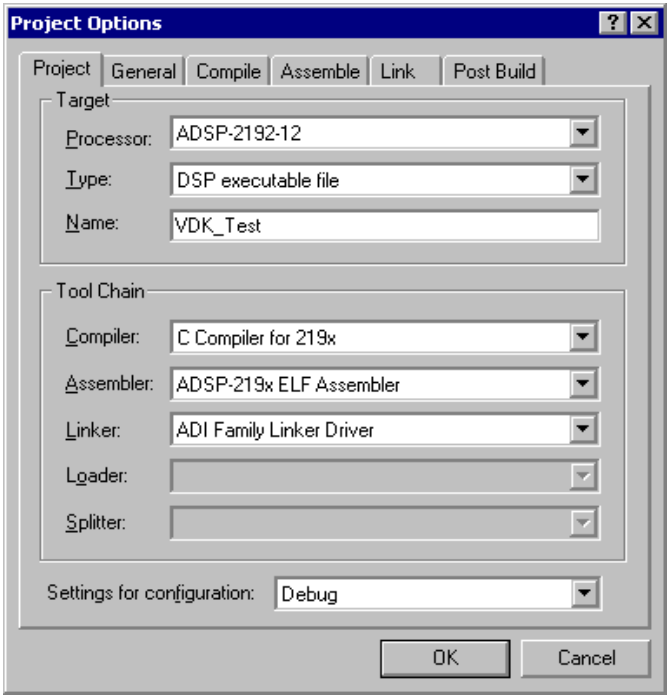

Figure 3-2. Project Options Dialog Box

5. Depending on the selected processor in step 4, you may be queried whether to include VisualDSP++ kernel instrumentation for this project (Figure 3-3). Click **Yes** to enable VDK support.

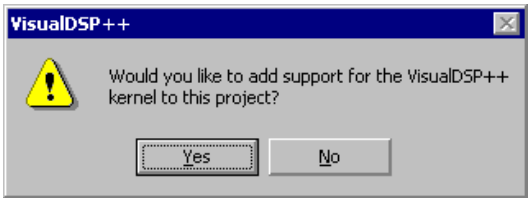

Figure 3-3. VisualDSP++ Kernel Message Box

6. The IDDE automatically generates the default VDK source files, `vdk.cpp` and `vdk.h`, and adds them to the project. A third file, `<project-name>.vdk`, is added to the project to enable VDK support in the IDDE.

`vdk.h` and `vdk.cpp` should not be modified, overwritten, or removed from the project.

7. Finally, add the appropriate VDK Linker Description File for the target processor. The **Project** tab, **Project** window in Figure 3-4 shows the added files.

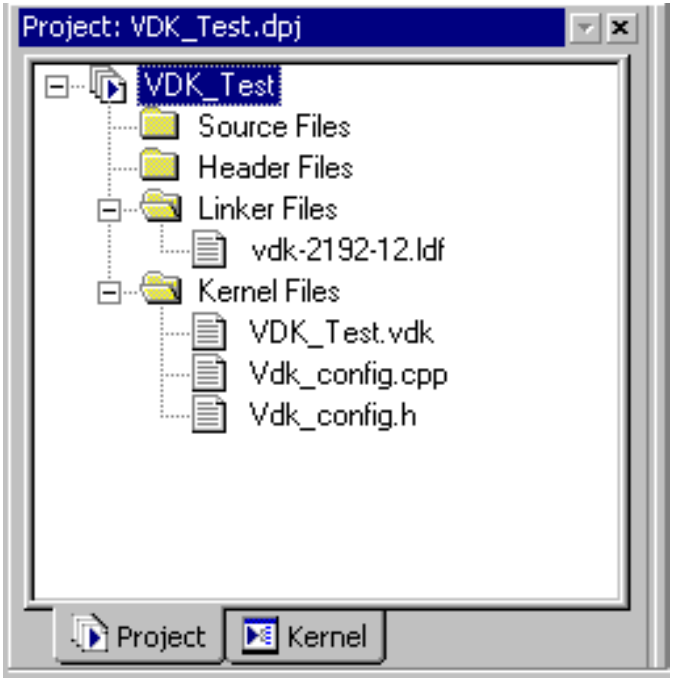

Figure 3-4. Project Window: Project Tab

8. The **Kernel** tab is visible at the bottom of the **Project** window. Left-click the tab to display the **Kernel** window (Figure 3-5 on page 3-8).

   In the **Kernel** window, you can add, modify, and delete kernel components: **System**, **Thread Types**, **Boot Threads**, **Round-Robin Properties**, **Semaphores**, **Event Bits**, **Events**, **Interrupts**, and **Device Drivers**. The IDDE automatically updates `vdk.cpp` and `vdk.h` to reflect any changes made in this window.

Step 2: Modify VDK System Parameters

To View and/or Modify System Parameters

1. Expand the **System** icon in the **Kernel** window by clicking the **Add** icon (Figure 3-5).

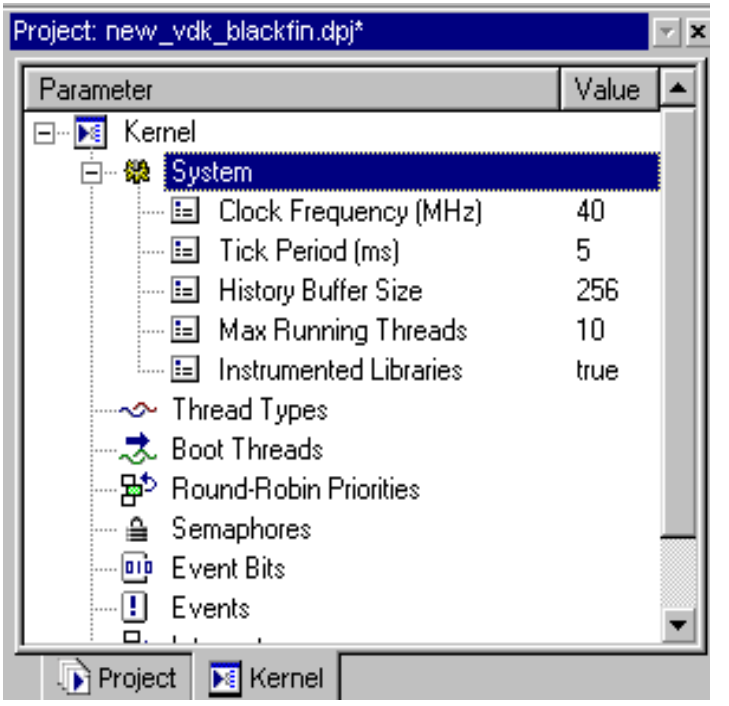

Figure 3-5. Expanded System Parameters Icon

The System component has the following properties:

Table 3-1. The System's Properties

Property	Description	Default	Note
Clock Frequency (MHz)	The clock frequency of your target processor.	40 MHz	Type float. The value is rounded to the nearest LSB of Clock Frequency; must be greater than zero.
Tick Period (ms)	The tick period of the system in milliseconds.	5 ms	Type float. The value is rounded to the nearest LSB of Clock Frequency; must be greater than one.
History Buffer Size	The number of entries in the history buffer.	256 4 words/entry allocated	Type int. Only valid when the Debug Enabled property is TRUE.
Max Running Threads	The upper limit of the number of simultaneously running threads in the system.	10	Type int. The value must be greater than zero.
Instrumented Libraries	Enables or disables the debugging features of the IDDE, such as history and state collection. Defines the `VDK_INCLUDE_INSTRUMENTATION_` macro.	TRUE	Type Boolean. Disable this feature to conserve memory and improve run-time efficiency.

Refer to "Round-Robin Scheduling" on page 4-11 and page 4-34 for more information about the system clock. Refer to "State History Window" on page 3-44 for more information about the history buffer.

Step 3: Add and Edit a Thread Type

To Add a New Thread Type to the Project

1. Right-click the **Thread Types** icon to display the context menu, then select **New Thread Type** from the menu (Figure 3-6).

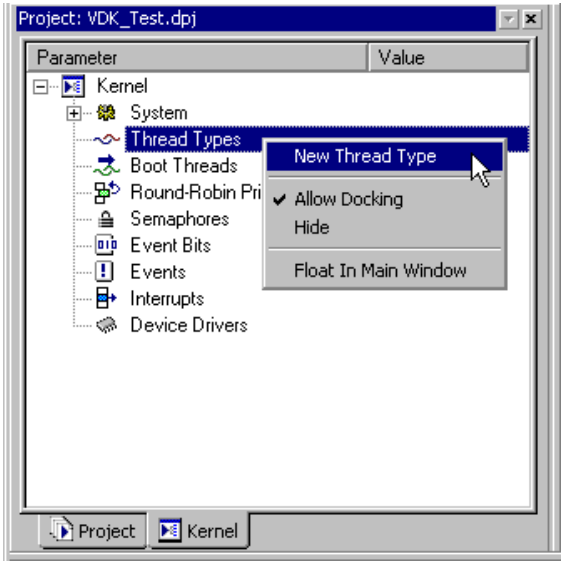

Figure 3-6. Thread Types: Context Menu

Refer to "Thread Types" on page 4-2 for information about thread types.

2. The **New Thread Type** dialog box appears on the screen (Figure 3-7 on page 3-11). Fill in the **Thread Type Name:**.

   A **Thread Type** name must be a valid C identifier (no special characters, such as spaces or hyphens, are allowed). You may change the **Source File:** and **Header File:** names in which the new **Thread Type** is defined.

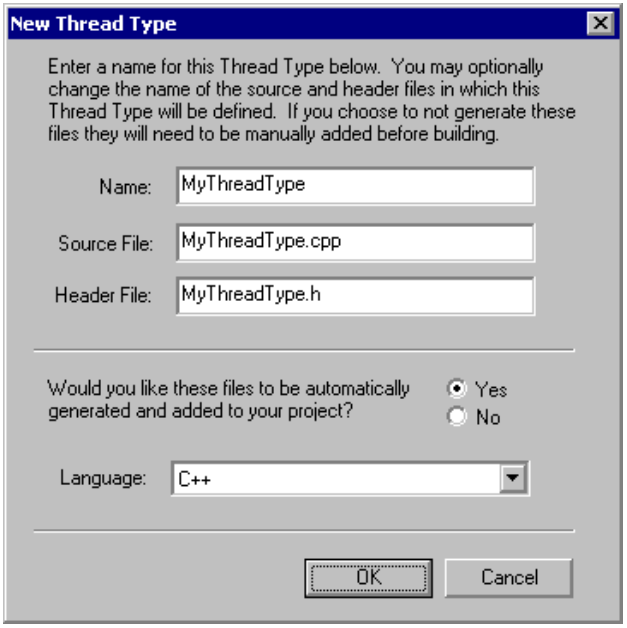
New Thread Type
Enter a name for this Thread Type below. You may optionally change the name of the source and header files in which this Thread Type will be defined. If you choose to not generate these files they will need to be manually added before building.
Name:
MyThreadType
Source File:
MyThreadType.cpp
Header File:
MyThreadType.h
Would you like these files to be automatically generated and added to your project?
Yes
No
Language:
C++
OK
Cancel

Figure 3-7. New Thread Type Dialog Box

3. Click **Yes** to automatically generate source code for this **Thread Type**. Selecting **Yes** allows you to choose the language in which the sources are generated. Your choices are: **C++**, **C**, and **Assembly**.

   If you want to use existing source code, click **No**. Then manually add the sources to your project. Note that the **Thread Type** name and file names must match.

4. Click **OK** to create the new **Thread Type**. If you have chosen to generate source code, the files are created and automatically added to the project.

If you create source files specifying filenames that are already in your project's directory, the existing files are overwritten.

To View/Modify a Thread Type's Properties

1. Expand the **Thread Type** icon in the **Kernel** window by clicking the **Add** icon (Figure 3-8). You can edit a property by clicking the left mouse button on the value of the property.

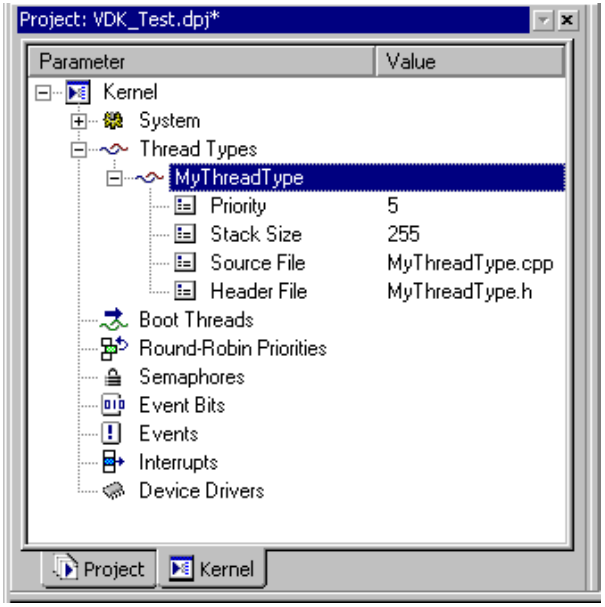

Figure 3-8. Expanded Thread Types Icon

The Thread Type component has the following properties:

Table 3-2. The Thread Types' Properties

Property	Description	Default	Notes
Priority	The initial priority of this Thread Type.	5	Type int. Limited to your processor's word size minus two. The highest priority is one.
Stack Size	The size of the stack in words. The value is set for each instance of this Thread Type.	255	Type int. Limited by system memory. The stack size must be greater than zero.
Source File	The source filename in which this Thread Type is implemented.	N/A	<*threadtype_name*>.cpp <*threadtype_name*>.c <*threadtype_name*>.asm
Header File	The header filename in which this Thread Type is defined.	N/A	<*threadtype_name*>.h

To Delete a Thread Type from the Project

1. Right-click the **Thread Type** icon you want to delete, then choose **Delete** from the context menu. Alternately, select the **Thread Type** icon and press the **Delete** key.

2. The IDDE prompts you to remove the source and header files for this **Thread Type** from the project. Click **Yes** to remove, or click **No** to leave the files in the project. Note that the files are not deleted from the project directory—files are removed from the **Project** window only.

Once you have deleted the **Thread Type** from the project, you should manually remove all references to the deleted **Thread Type** from your code.

Step 4: Add and Edit a Boot Thread

To Add a New Boot Thread to the Project

1. Right-click the **Boot Threads** icon and select **New Boot Thread** from the context menu (Figure 3-9). A new **Boot Thread** is created.

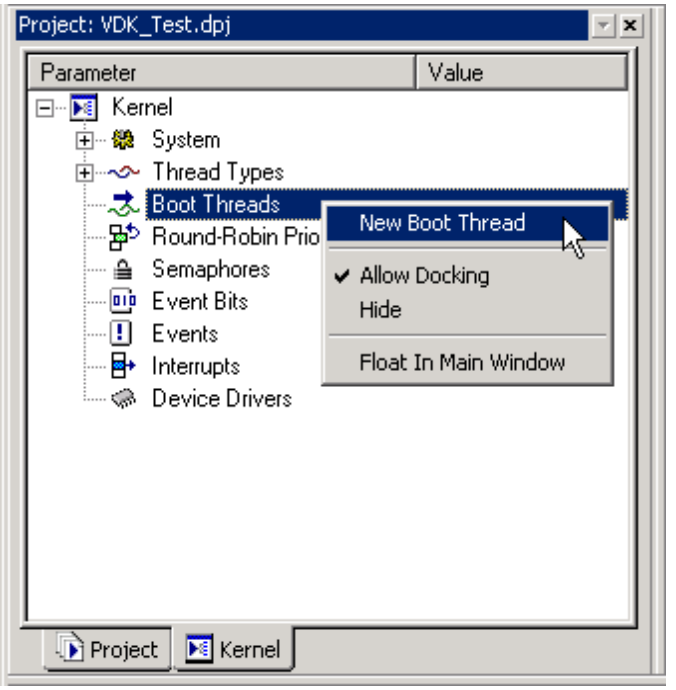

Figure 3-9. Boot Thread: Context Menu

2. Enter a name for the new **Boot Thread** and press **Enter**.

To View/Modify a Boot Thread's Properties

1. Expand the **Boot Thread** icon in the **Kernel** window by clicking the **Add** icon. You can edit a property by clicking the left mouse button on the value of the property.

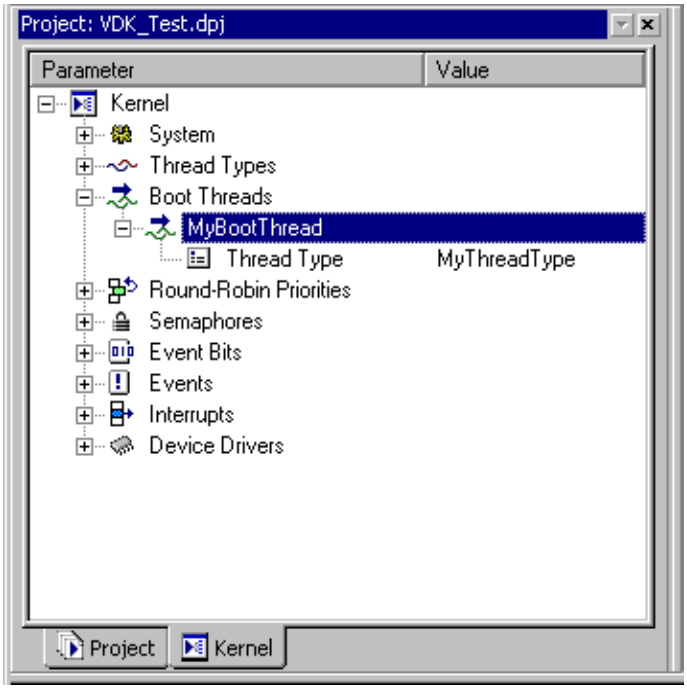

Figure 3-10. Expanded Boot Threads Icon

Multiple boot threads of the same thread type are allowed. Refer to page 4-2 for information about boot threads.

To Rename a Boot Thread

1. Click the right mouse button on the **Boot Thread** to display the context menu, then choose **Rename** from the menu. Alternately, select the **Boot Thread** icon and press **F2**. Fill in the **Boot Thread Name.**

2. Press **Enter** to accept the change, press **ESC** to cancel the change.

To Delete a Boot Thread from the Project

1. Click the right mouse button on the **Boot Thread** to display the context menu, then choose **Delete** from the menu. Alternately, select the **Boot Thread** icon and press the **Delete** key.

Once you have removed the **Boot Thread** from the project, you should manually remove all references to the deleted **Boot Thread** from your code.

To Change the Order in which Boot Threads are Created

You can change the order in which the Boot Threads are created by dragging and dropping the **Boot Thread** icons.

1. Left-click and drag the icon of the **Boot Thread** that you want to reorder.

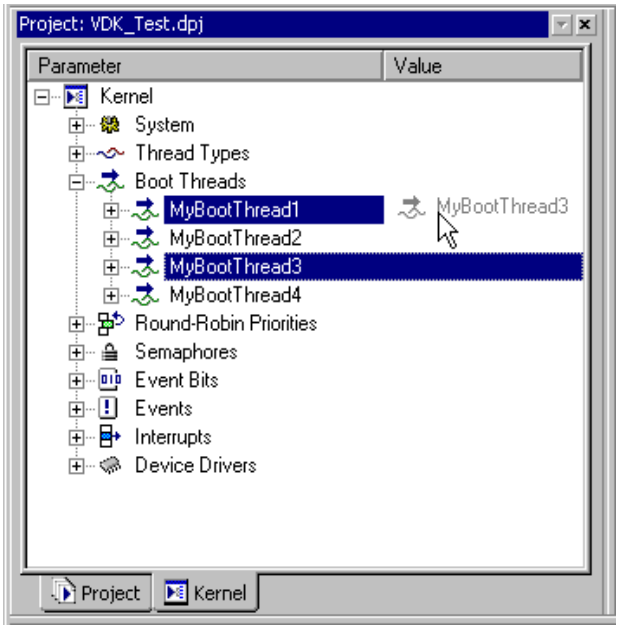

Figure 3-11. Moving Boot Thread Objects

2. Release the left mouse button when the **Boot Thread** is in the correct location.

   In the example above (Figure 3-11), **MyBootThread3** is being moved to the first position in the list. **MyBootThread3** is to be created first when the kernel boots. This determines the order in which the **MyBootThread3**'s constructor is called. However, the boot thread runs in the order that is based on its priority.

Step 5: Add and Edit a Round-Robin Priority

To Add a Round-Robin Priority to the Project

1. Right-click the **Round-Robin Priorities** icon to display the context menu, and then select **New Priority** from the menu (Figure 3-12). A new **Round-Robin Priority** is created.

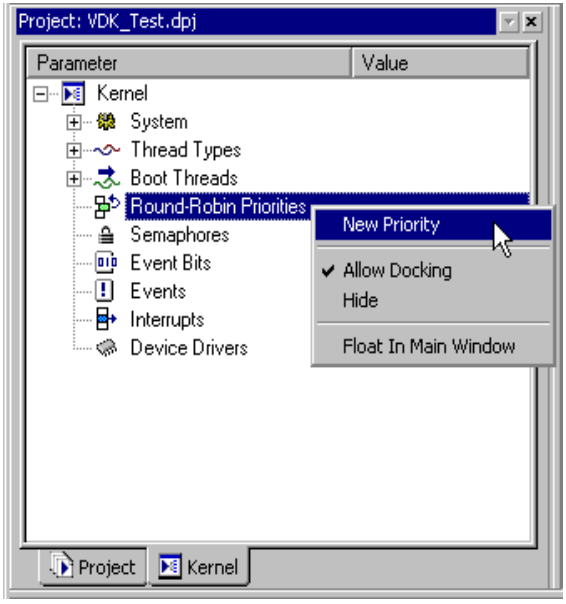

Figure 3-12. Round-Robin Priority: Context Menu

For information about round-robin priorities, see "Round-Robin Scheduling" on page 4-11.

2. From the drop-down list, select the priority that you want to make round-robin. Press **Enter** to accept the priority (Figure 3-13).

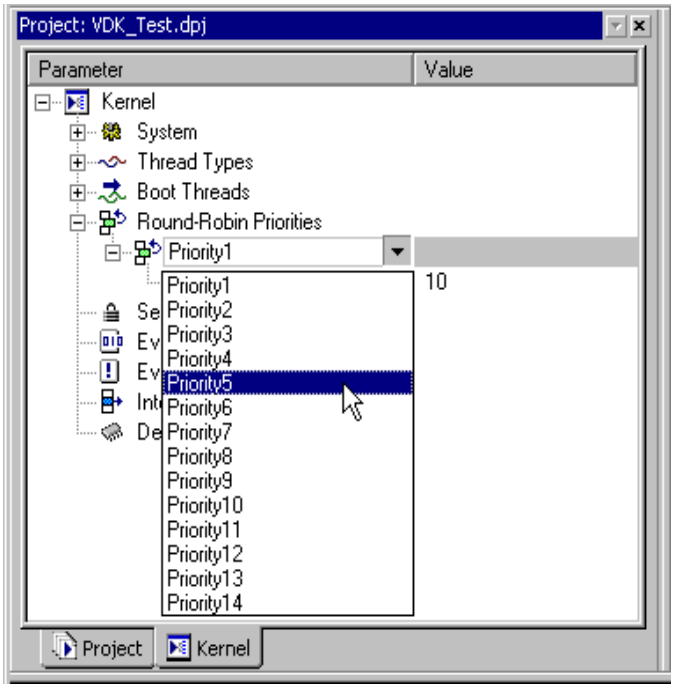

Figure 3-13. Selecting a Round-Robin Priority

To View/Modify a Round-Robin's Priorities

1. Expand the **Round-Robin Priorities** icon in the **Kernel** window by clicking the **Add** icon. You can edit a property by clicking the left mouse button on the value of the property (Figure 3-14).

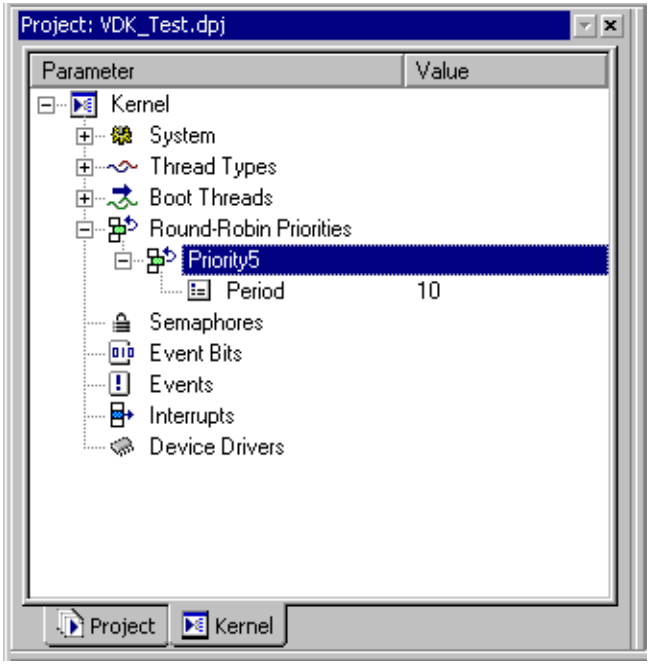

Figure 3-14. Expanded Round-Robin Properties Icon

The Round-Robin Priority component has the following properties:

Table 3-3. The Round-Robin Priorities

Property	Description	Default	Notes
Period	The period in Ticks for this priority.	10	Type int. Must be greater than zero.

To Modify a Round-Robin Priority

1. Click the right mouse button on the **Round-Robin Priority** to display the context menu, and then choose **Rename** from the menu. Alternately, select the **Round-Robin Priority** icon and press **F2**. Select the new name from the drop-down list of predefined names.

2. Press **Enter** to accept the change, press **ESC** to cancel.

To Delete a Round-Robin Priority from the Project

1. Click the right mouse button on the **Round-Robin Priority** to display the context menu, and choose **Delete** from the menu. Alternately, select the **Round-Robin Priority** icon and press the **Delete** key.

Step 6: Add and Edit a Semaphore

To Add a Semaphore to the Project

1. Right-click the **Semaphores** icon to display the context menu, select **New Semaphore** from the menu (Figure 3-15). A new **Semaphore** is created.

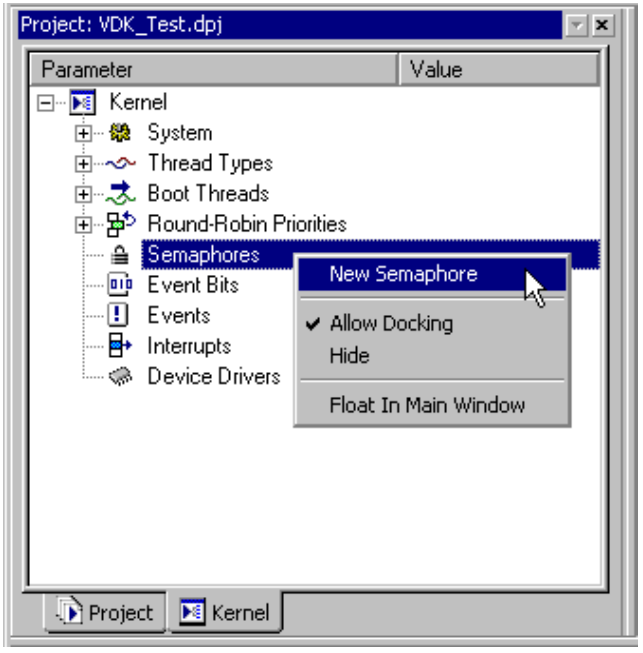

Figure 3-15. Semaphores: Context Menu

2. Enter a name for the new **Semaphore** and press **Enter**. A **Semaphore** name must be a valid C identifier.

   Refer to page 4-16 for information about semaphores.

To View/Modify a Semaphore's Properties

1. Expand the **Semaphore** icon in the **Kernel** window by clicking the **Add** icon. You can select and edit a property by clicking the left mouse button on the value of the property (Figure 3-16).

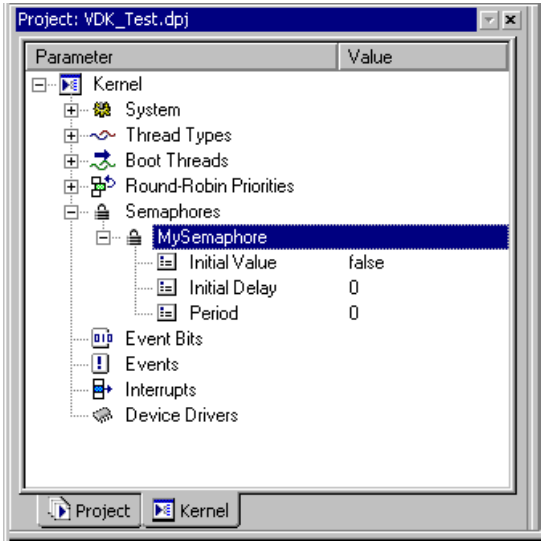

Figure 3-16. Expanded Semaphores Icon

The Semaphore component has the following properties:

Table 3-4. The Semaphores' Properties

Property	Description	Default	Notes
Initial Value	The value to which this semaphore is set when the kernel boots.	FALSE	Type Boolean.
Initial Delay	The initial delay time for this Semaphore in Ticks	0	Type int. You must set both values to make a Semaphore periodic.
Period	The period in Ticks for which this Semaphore will become signaled.	0	

To Rename a Semaphore

1. Click the right mouse button on the **Semaphore** icon to display the context menu, and then choose **Rename** from the menu. Alternately, select the **Semaphore** icon and press **F2**. Enter the new name for the **Semaphore**. The name must be a valid C identifier.
2. Press **Enter** to accept the change, press **ESC** to cancel the change.

You should manually update all references to the **Semaphore** of the modified name in your code.

To Delete a Semaphore from the Project

1. Click the right mouse button on the **Semaphore** to display the context menu, and click **Delete** from the menu. Alternately, select the **Semaphore** icon and press the **Delete** key.

Once you have removed the **Semaphore** from the project, you should manually remove all references to the deleted **Semaphore** from your code.

Step 7: Add and Edit an Event Bit

To Add an Event Bit to the Project:

1. Right-click the **Event Bits** icon and select **New Event Bit** from the context menu. A new **Event Bit** is created (Figure 3-17).

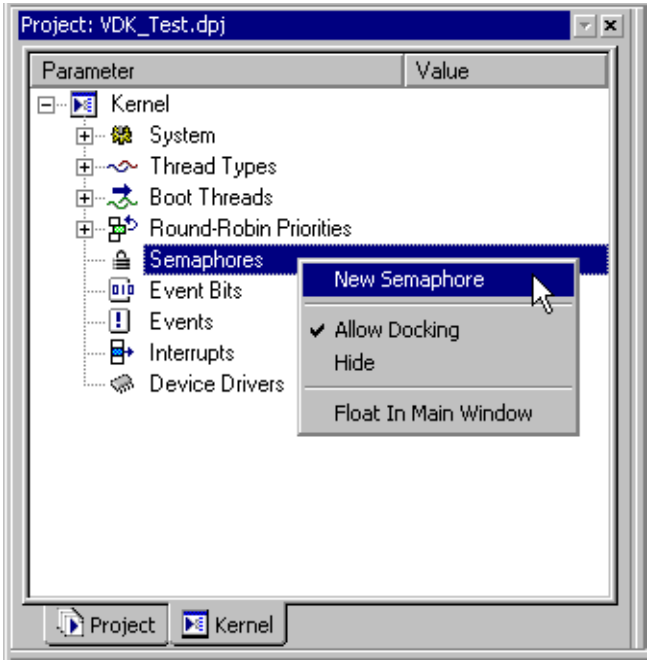

Figure 3-17. Event Bits: Context Menu

2. Enter a name for the new **Event Bit** and press **Enter**. An **Event Bit** name must be a valid C identifier.

   Refer to page 4-21 for information about the event bits.

To View/Modify an Event Bit's Properties

1. Expand the **Event Bit** icon in the **Kernel** window by clicking the **Add** icon. You can select and edit a property by clicking the left mouse button on the value of the property (Figure 3-18).

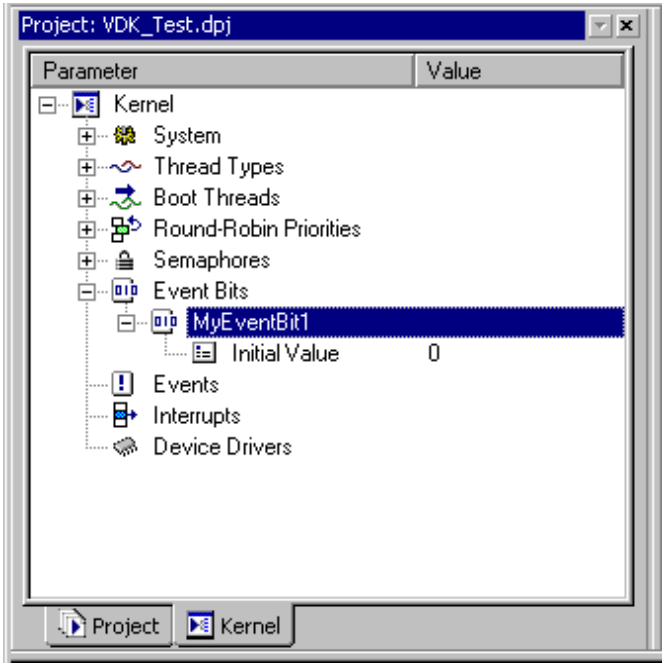

Figure 3-18. Expanded Event Bits Properties Icon

The Event Bit component has the following properties:

Table 3-5. The Event Bits' Properties

Property	Description	Default	Notes
Initial Value	The value to which this event bit is set when the kernel boots.	0	Type int.

To Rename an Event Bit

1. Click the right mouse button on the **Event Bit** to display the context menu, and then select **Rename** from the menu. Alternately, select the **Event Bit** icon and press **F2**. An **Event Bit** name must be a valid C identifier.

2. Once the new name has been entered, press **Enter** to accept the change or **ESC** to cancel.

You should manually update all references to the **Event Bit** of the modified name in your code.

To Delete an Event Bit from the Project

1. Click the right mouse button on the **Event Bit** to display the context menu, and click **Delete** from the menu. Alternately, select the **Event Bit** icon and press the **Delete** key.

Once you have deleted the **Event Bit** from the project, you should manually remove all references to the **Event Bit** from your code.

Step 8: Add and Edit an Event

To Add an Event to the Project

1. Right-click the **Events** icon, select **New Event** from the context menu (Figure 3-19).

   Note that there must be at least one **Event Bit** defined before an **Event** can be created. A new **Event** is created.

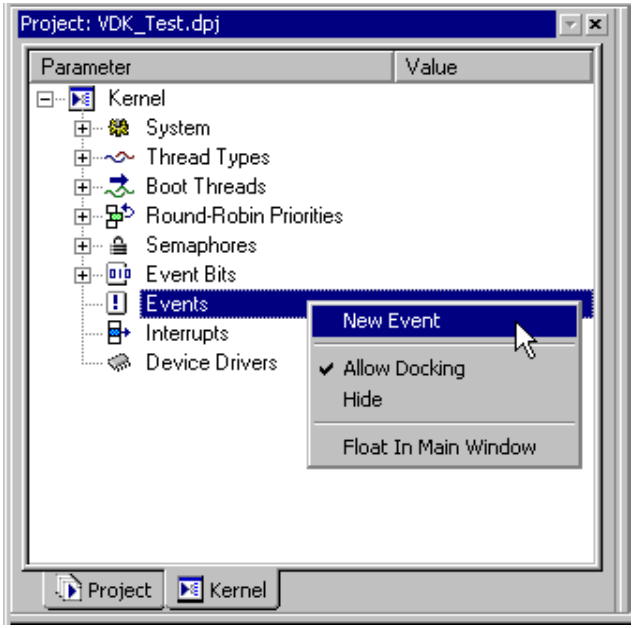

Figure 3-19. Events: Context Menu

2. Enter a name for the new **Event** and press **Enter**. An Event name must be a valid C identifier.

   For information about events, refer to "Events and Event Bits" on page 4-21.

To View/Modify an Event's Properties

1. Expand the **Event** icon in the **Kernel** window by clicking the **Add** icon. You can select and edit a property by clicking the left mouse button on the value of the property (Figure 3-20).

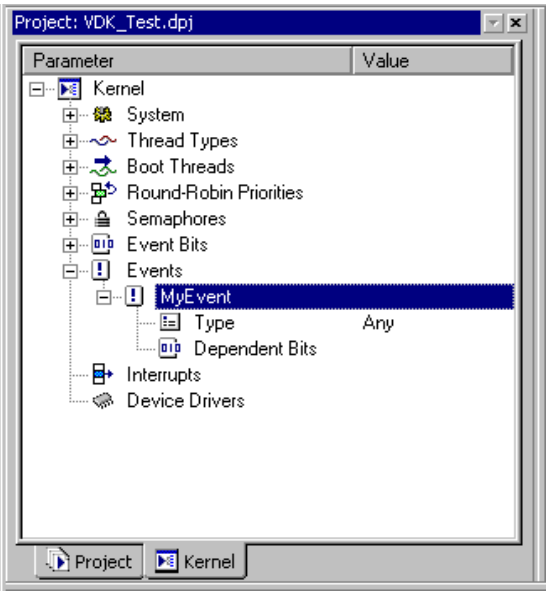

Figure 3-20. Expanded Events Properties Icon

The Event component has the following properties:

Table 3-6. The Events' Properties

Property	Description	Default	Notes
Type	The condition upon which the event becomes signaled.	Any	Any is selected: the event becomes signaled when any one of the event bits' conditions is met. All is selected: the event becomes signaled only when all of the event bits' conditions are met.

To Rename an Event

1. Click the right mouse button on the **Event** to display the context menu, and select **Rename** from the menu. Alternately, click the **Event** icon and press **F2**. An Event name must be a valid C identifier.

2. Press **Enter** to accept the change, press **ESC** to cancel the change.

You must manually update all references to the **Event**'s modified name in your code.

To Delete an Event from the Project

1. Click the right mouse button on the **Event** to display the context menu, and click **Delete** from the menu. Alternately, select the **Event** icon and press the **Delete** key.

Once you have deleted the **Event** from the project, you should manually remove all references to the **Event** from your code.

To Add Event Bits to an Event

An Event must have at least one Event Bit and an associated condition.

1. Right-click the **Dependent Bits** icon to display the context menu, and choose **Add Dependency** from the menu (Figure 3-21).

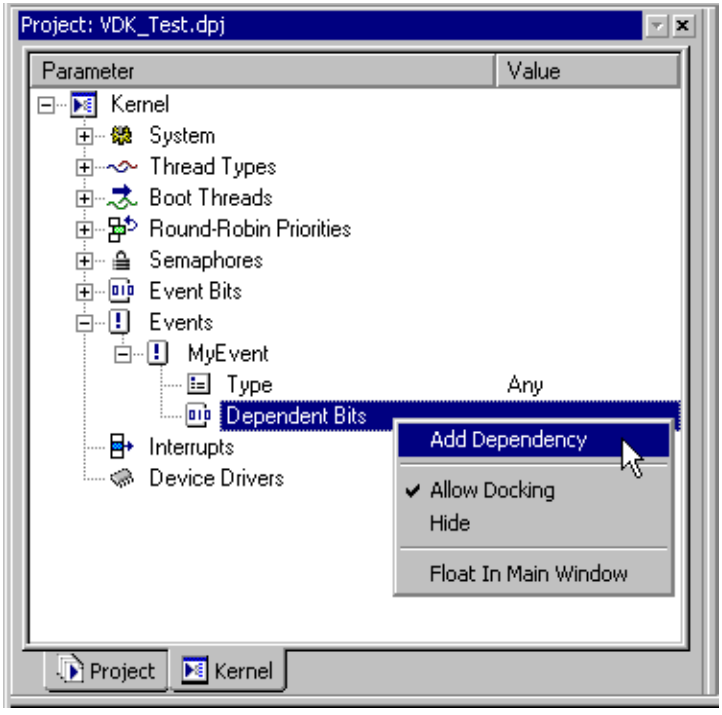

Figure 3-21. Events: Context Menu

2. Choose the desired **Event Bit** from the drop-down list, and then press **Enter** to accept the new **Event Bit** dependency (Figure 3-22).

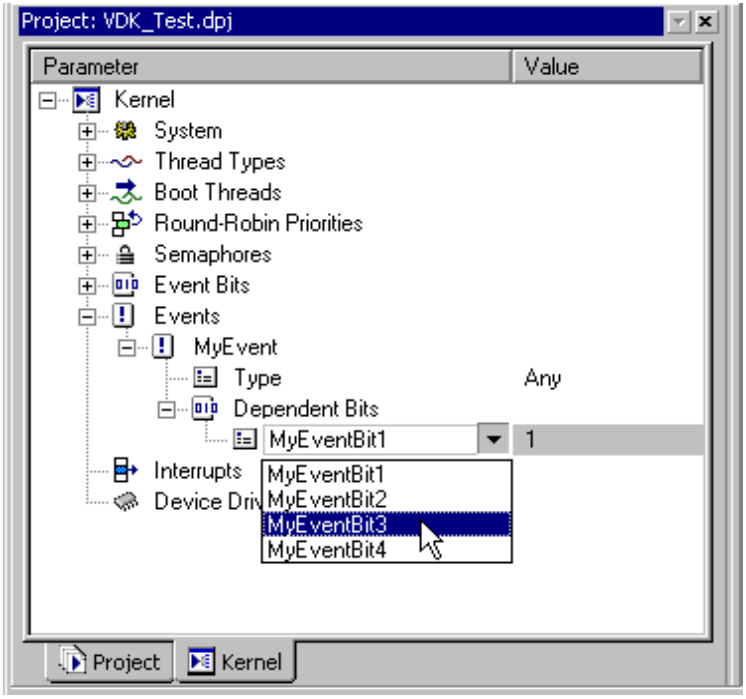

Figure 3-22. Events: Dependency Drop-Down List

To View/Modify a Dependent Event Bit's Condition

1. You may edit a dependent **Event Bits** condition by clicking the left mouse button on the value of the condition. Possible values are: 1 or 0 (Figure 3-23).

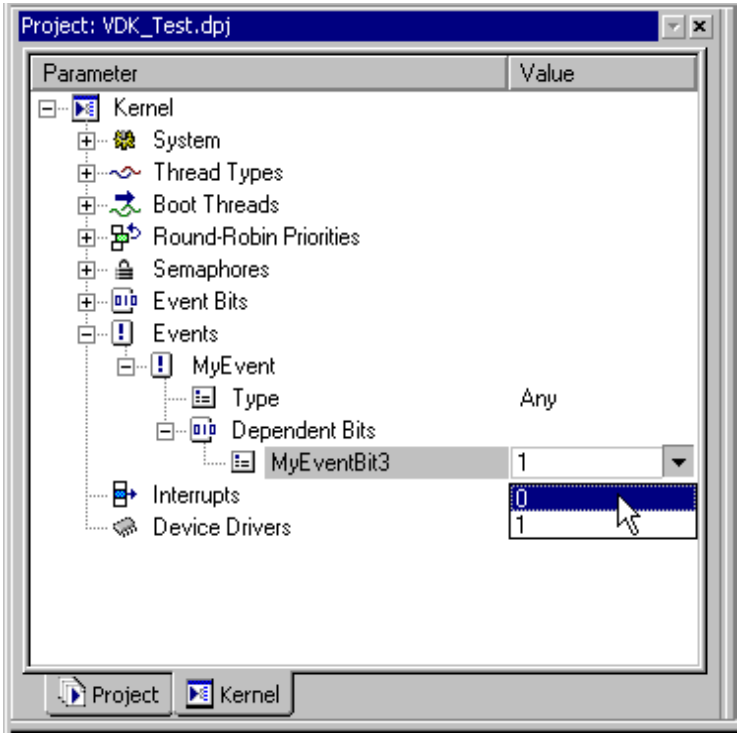

Figure 3-23. Evens: Dependency Condition Drop-Down List

Step 9: Add and Edit an Interrupt

To Add an Interrupt to the Project

1. Right-click the **Interrupts** icon and select **New Interrupt** from the context menu (Figure 3-24).

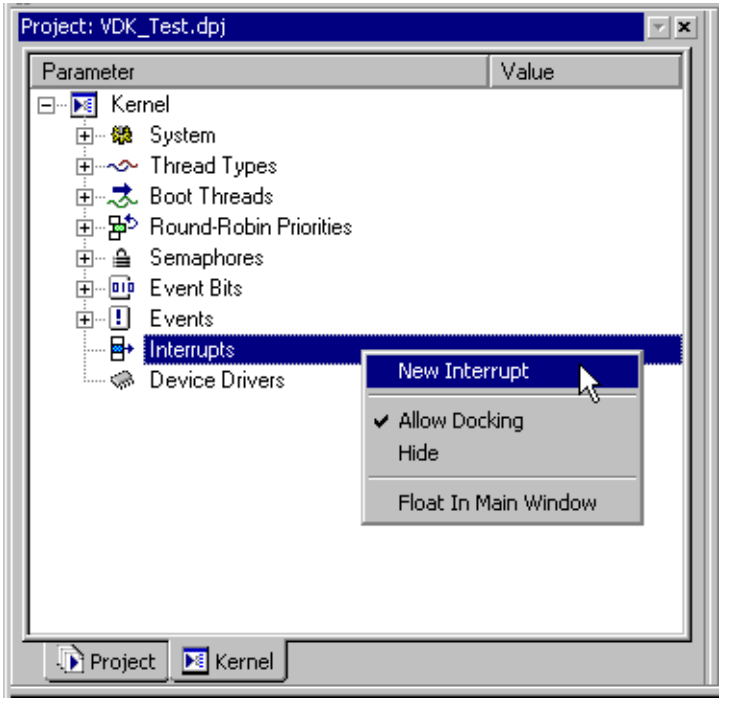

Figure 3-24. Interrupts Context Menu

Refer to "Interrupt Service Routines" on page 4-30 for information about interrupts.

2. The **New Interrupt** dialog box appears (Figure 3-25). In the **Name:** field, select the desired **Interrupt** from the drop-down list. You can optionally change the **Source File:** name in which the new **Interrupt** is defined.

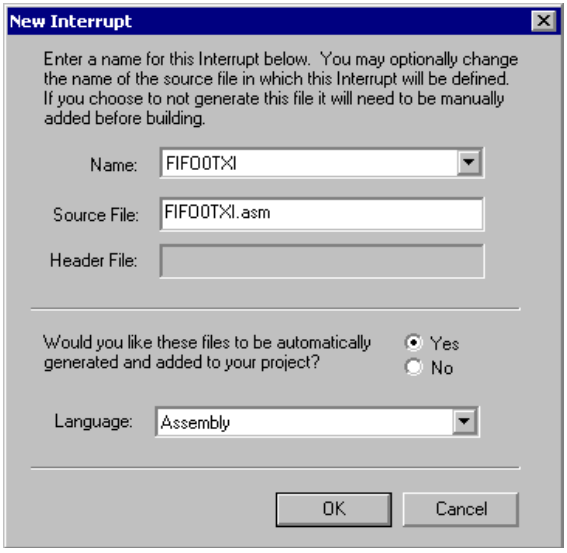
New Interrupt
Enter a name for this Interrupt below. You may optionally change the name of the source file in which this Interrupt will be defined. If you choose to not generate this file it will need to be manually added before building.
Name:
FIFOOTXI
Source File:
FIFOOTXI.asm
Header File:
Would you like these files to be automatically generated and added to your project?
Yes
No
Language:
Assembly
OK
Cancel

Figure 3-25. New Interrupt Dialog Box

3. To automatically generate a source code for this **Interrupt**, click **Yes**. Note that **Interrupts** are generated in **Assembly** language only.

   If you wish to use the existing source code, click **No**. Then manually add the source to your project. Note that the **Interrupt** name and source file name must match.

4. Click **OK** to create the new **Interrupt**. If you have chosen to generate a source code, the source file is created and automatically added to the project.

If you create a source file specifying the filename that is already in your project's directory, the existing file is overwritten.

To View/Modify an Interrupt's Properties

1. Expand the **Interrupt** icon in the **Kernel** window by clicking the **Add** icon. You can edit a property by clicking the left mouse button on the value of the property (Figure 3-26).

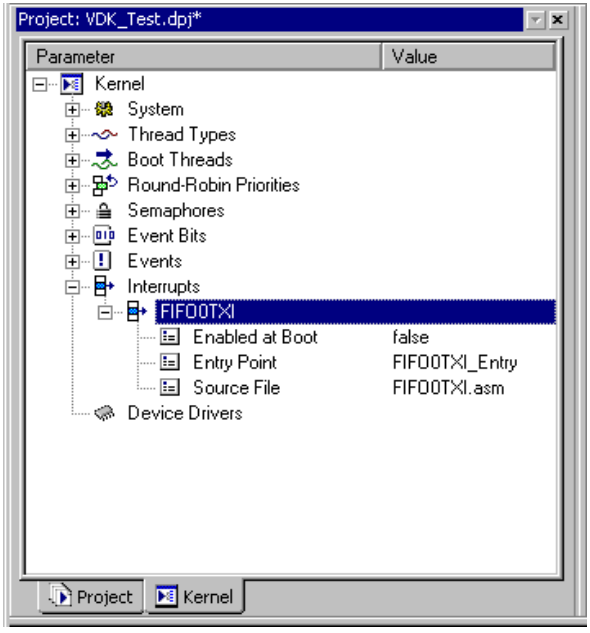

Figure 3-26. Expanded Interrupts Icon

The Interrupt component has the following properties:

Table 3-7. The Interrupts' Properties

Property	Description	Default	Notes
Enabled at Boot	Controls whether this interrupt is masked or not when the kernel boots.	FALSE	Type Boolean. You manually enable the interrupt at runtime if it is not enabled at kernel boot.
Entry Point	The entry point label for this Interrupt Service Routine.	<*interrupt_name*>_Entry	
Source File	The source filename in which this Interrupt is implemented.	<*interrupt_name*>.asm	

To Delete an Interrupt from the Project

1. Click the right mouse button on the **Interrupt** to display the context menu, and click **Delete** from the menu. Alternately, select the **Interrupt** icon and press the **Delete** key.

2. The IDDE prompts you to remove the source file for this **Interrupt** from the project. Click **Yes** to remove the **Interrupt,** or click **No** to leave it in the project. Note that the file is not deleted from the project directory —the file is removed from the **Project** window only.

Once you have deleted the **Interrupt** from the project, you should manually remove all references to that **Interrupt** from your code.

Step 10: Add and Edit a Device Driver

To Add a Device Driver to the Project

1. Right-click the **Device Drivers** icon to display the context menu, select **New Device Driver** from the menu (Figure 3-27).

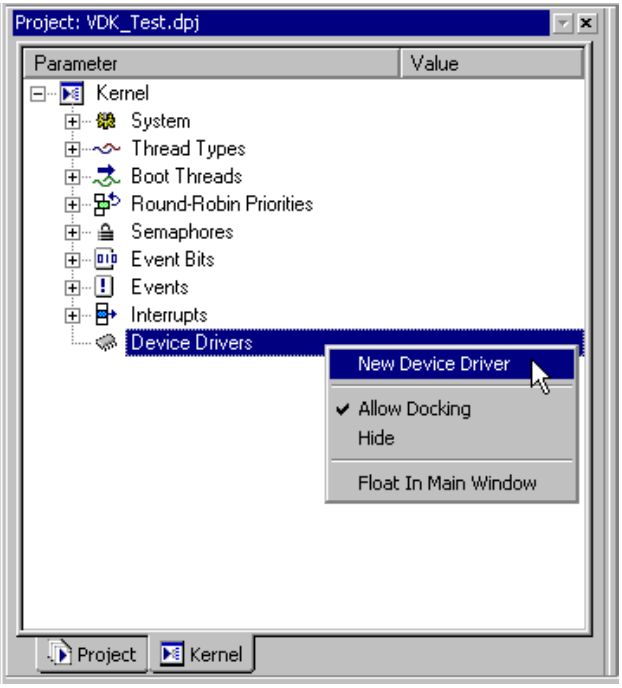

Figure 3-27. Device Drivers: Context Menu

Refer to "Device Drivers" on page 4-35 for information about device drivers.

2. The **New Device Driver** dialog appears on the screen. Fill in the **Device Driver Name:** field. A **Device Driver** name must be a valid C identifier. You may change the **Source File:** and **Header File:** names in which the new **Device Driver** is defined (Figure 3-28).

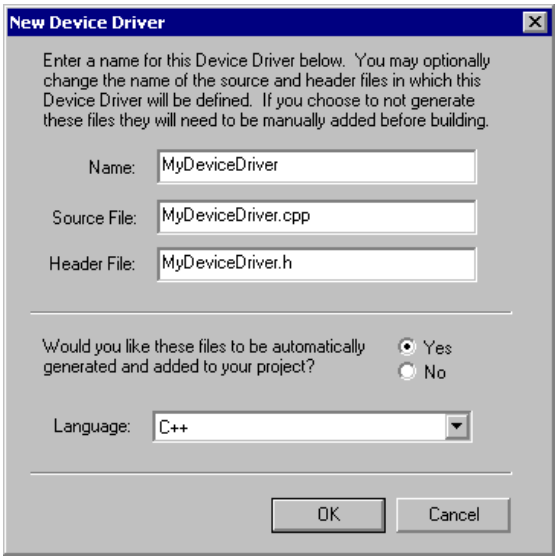
New Device Driver
Enter a name for this Device Driver below. You may optionally change the name of the source and header files in which this Device Driver will be defined. If you choose to not generate these files they will need to be manually added before building.
Name:
MyDeviceDriver
Source File:
MyDeviceDriver.cpp
Header File:
MyDeviceDriver.h
Would you like these files to be automatically generated and added to your project?
Yes
No
Language:
C++
OK
Cancel

Figure 3-28. New Device Driver Dialog Box

3. To automatically generate the source code for this **Device Driver**, click **Yes**. Selecting **Yes** allows you to choose the language in which the source is generated. Possible choices are: **C++** and **C**.

   If you wish to use the existing source code, click **No**. Then manually add the source to your project. Note that the **Device Driver** name and file names must match.

   If you create source files specifying filenames that are already in your project's directory, the existing files are overwritten.

4. Click **OK** to create the new **Device Driver**. If you have chosen to generate a source code, the files are created and added to the project.

To View/Modify a Device Driver's Properties

1. Expand the **Device Driver** icon in the **Kernel** window by clicking its Add icon. You can edit a property by clicking the left mouse button on the value of the property (Figure 3-29).

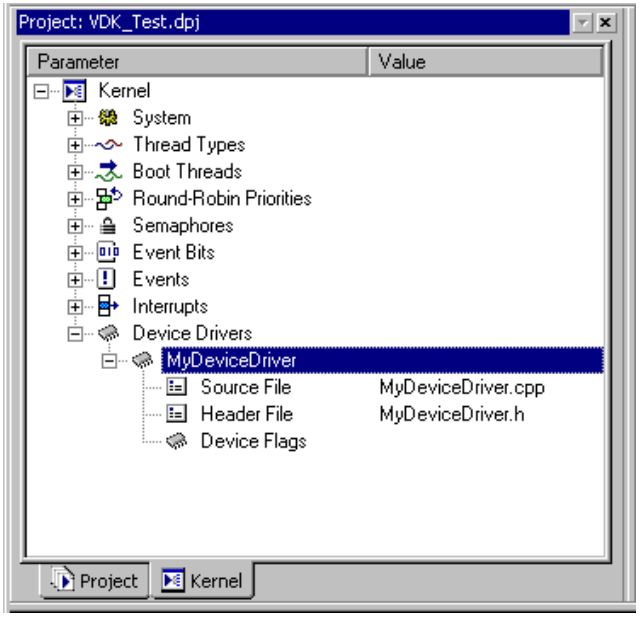

Figure 3-29. Expanded Device Drivers Icon

The Device Driver component has the following properties:

Table 3-8. The Device Drivers' Properties

Property	Description	Default
Source File	The source filename in which this Driver is implemented.	`<driver_name>.cpp`
Header File	The header filename in which this Device Driver is defined.	`<driver_name>.h`

To Delete a Device Driver from the Project

1. Click the right mouse button on the **Device Driver** to display the context menu, and choose **Delete** from the menu. Alternately, select the **Device Driver** icon and press the **Delete** key.

2. The IDDE prompts you to remove the source and header files for this **Device Driver** from the project. Click **Yes** to remove the files, or click **No** to leave the files in the project. Note that the files are not deleted from the project directory—files are removed from the **Project** window only.

Once you have deleted the **Device Driver** from the project, you should manually remove all references to that **Device Driver** along with all associated **Device Flags** from your code.

To Add Device Flags to a Device Driver

A Device Driver can have associated Device Flags.

1. Click the right mouse button on the **Device Flags** icon to display the context menu, and choose **Add Device Flag** from the menu (Figure 3-30).

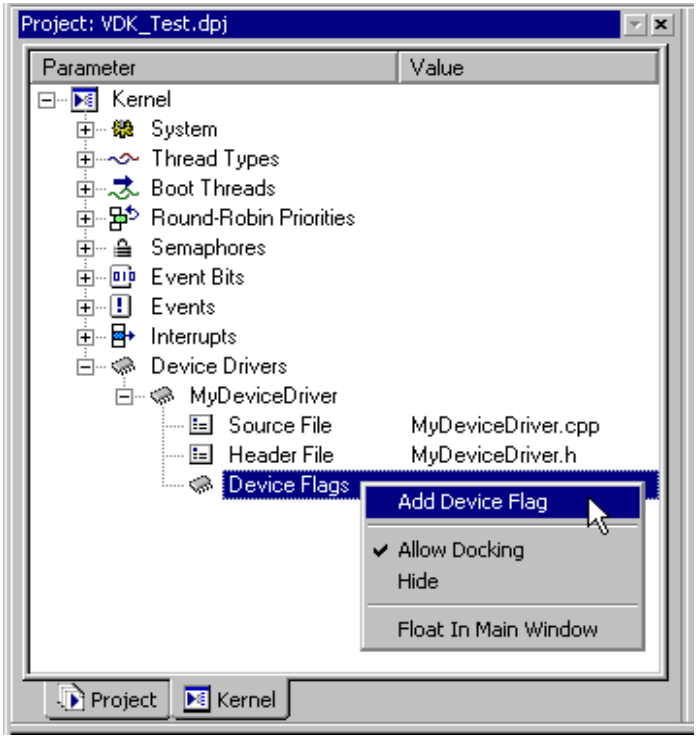

Figure 3-30. Device Flags: Context Menu

2. Give the **Device Flag** a name and press **Enter** to create the new flag. A **Device Flag** name must be a valid C identifier.

   Device Flags must be unique to the system, meaning that two **Device Drivers** cannot define the same **Device Flag** name (Figure 3-31).

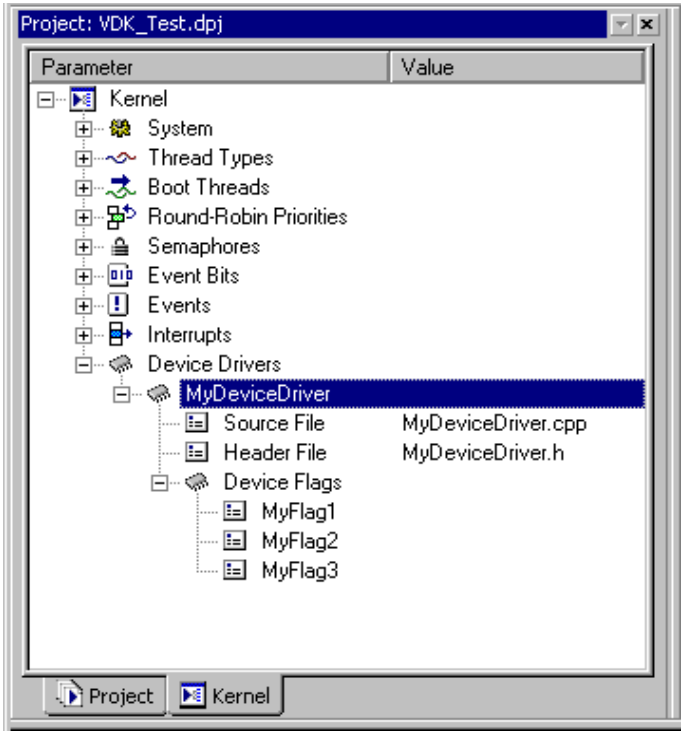

Figure 3-31. Expanded Device Flags Icon

Debugging a VDK Project

Debugging embedded software is a difficult task. To help offset the initial difficulties present in debugging your VDK-enabled projects, the kernel offers special instrumented builds. This section documents information that can help to debug your VDK software. For more information on this topic, see "Debugged Control Structures" on page 2-2.

Instrumented Build Information

When building a VDK project, you have an option to include instrumentation in your executable. An instrumented build differs from a release or non-instrumented build because the build includes an extra code for thread statistic logging. In addition, an instrumented build creates a circular buffer of important system events. The extra logging introduces slight overhead in thread switches and certain API calls, but helps you to trace system activities.

State History Window

The VDK logs user-defined events and certain system state changes in a circular buffer. An event is logged in the history buffer with a call to `LogHistoryEvent()`. The call to `LogHistoryEvent()` logs four data values: the `ThreadID` of the calling thread, the tick when the call had happened, the enumeration, and a value that is specific to the enumeration. Enumerations less than `0` are reserved for use by the VDK:

```
typedef enum
{
   VDK_kThreadCreated = INT_MIN,
   VDK_kThreadDestroyed,
   VDK_kSemaphorePosted,
   VDK_kSemaphorePended,
   VDK_kEventBitSet,
   VDK_kEventBitCleared,
   VDK_kEventPended,
```

```
    VDK_kDeviceFlagPended,
    VDK_kDeviceFlagPosted,
    VDK_kDeviceActivated,
    VDK_kThreadTimedOut,
    VDK_kThreadSwitched
} VDK_HistoryEnum;
```

Using the history log, the IDDE displays a graph of running threads and system state changes. The State History window, described in Figure 3-32, displays the history buffer plots. Note that the history buffer information is updated only on halt.

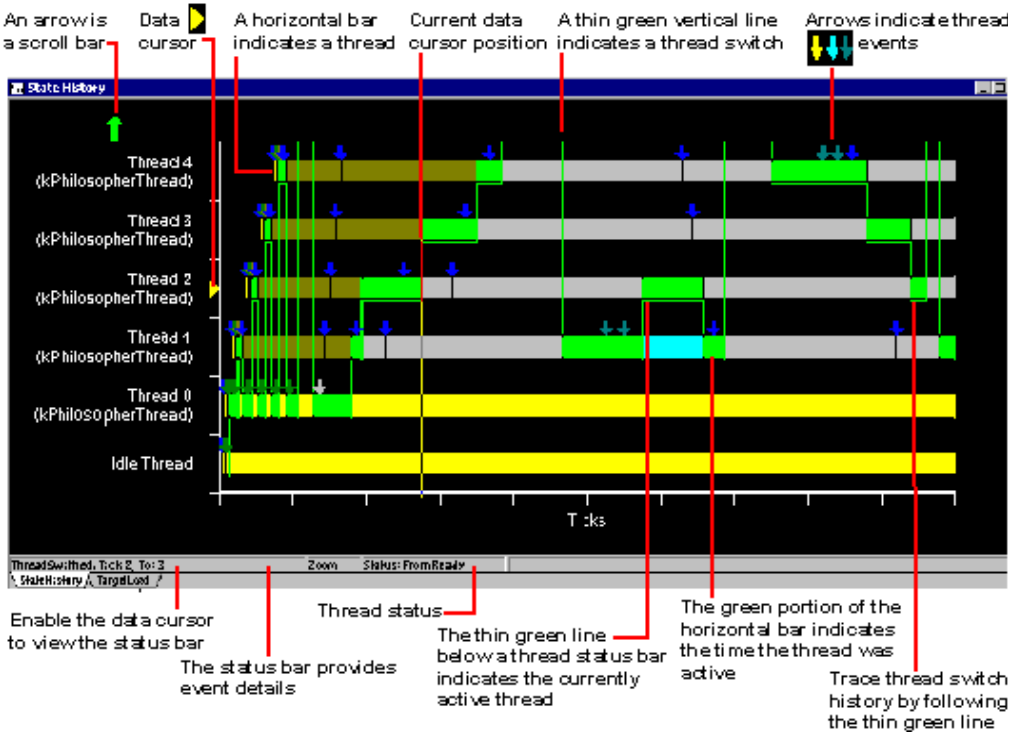
An arrow is a scroll bar
Data cursor
A horizontal bar indicates a thread
Current data cursor position
A thin green vertical line indicates a thread switch
Arrows indicate thread events
State History
Thread 4 (kPhilosopherThread)
Thread 3 (kPhilosopherThread)
Thread 2 (kPhilosopherThread)
Thread 1 (kPhilosopherThread)
Thread 0 (kPhilosopherThread)
Idle Thread
T :ks
Enable the data cursor to view the status bar
The status bar provides event details
Thread status
The thin green line below a thread status bar indicates the currently active thread
The green portion of the horizontal bar indicates the time the thread was active
Trace thread switch history by following the thin green line

Figure 3-32. System State History Window*

Thread status appears as horizontal bars, and thread events appear as arrows above the horizontal bars. Status bars and event symbols are color-coded, based on thread status and event type.

All events of the same type are drawn in the same color. The thin green line below the thread status bar indicates the currently active thread.

---

* Multiple events within a tick are spaced evenly for the display purposes.

When a thread is switched, the green line is drawn vertically to the next active thread. Trace the thread switch history by following the thin green line.

Thread Status and Thread Event Windows

Right-click on the plot and choose **Status Legend** (Figure 3-33) or **Event Legend** (Figure 3-34) to display the legends (shown below) in the State History window.

Figure 3-33. Thread Status Legend

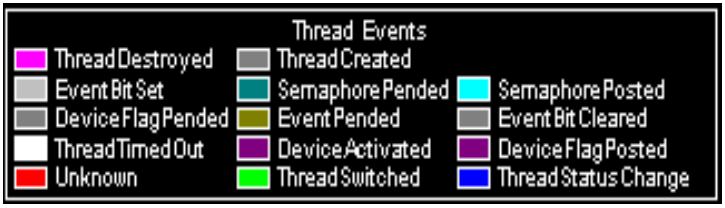

Figure 3-34. Thread Event Legend

Refer to "VDK Status Window" on page 3-50 for more information about VDK thread status.

Window Operations

The state history status bar (bottom of plot) shows the event's details and thread's status. Event details include the event type, the tick when the event occurred, and an event value. The value for a thread-switched event indicates the thread being switched in or switched out. The yellow triangle to the right of the thread name indicates the currently active thread.

Right-click on the plot and choose **Data Cursor** to activate the data cursor, which is used to display event and thread status details. Based on the event that occurred, the thread status changes. Press the keyboard's right arrow key or left arrow key to move to the next or previous event. When the data cursor hits a thread switch event, it moves to the thread being switch in.

You can zoom in on a region to examine that area in more detail. Hold the left mouse button down while dragging the mouse to create a selection box. Then release the mouse button to expand the plot. To restore the plot to its original scale, right-click on the plot and choose **Reset Zoom**.

Target Load Graph

Instrumented VDK builds allow you to analyze the average load of the processor over a period of time. Although the load calculation is not exact, the load graph helps you to estimate the utilization level of the processor. Note that the information used in the graph is updated at halt.

The target load graph shown in Figure 3-35 displays the percent of time the target spent in the idle thread.

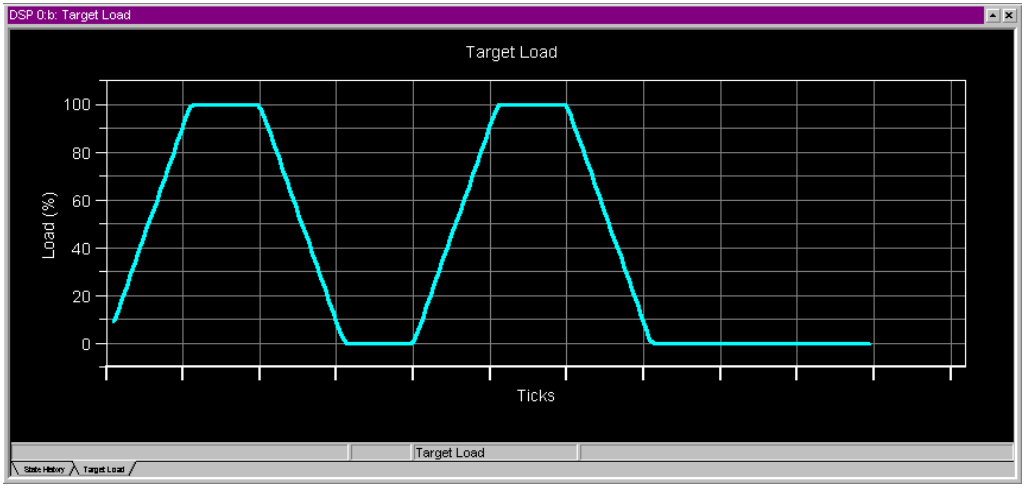

Figure 3-35. Target Load Window

A load of 0% means VDK spent all of its time in the idle thread. A load of 100% means the target did not spend any time in the idle thread. Load data is processed using a moving window average.

The load percentage is calculated for every clock tick, and all the ticks are averaged. The following formula is used to calculate the percentage of utilization for every clock tick:

Load = 1 - (# of times idle thread ran this tick) / (# of threads run this tick)

VDK Status Window

Besides history and processor load information, an instrumented build collects a thread's statistics, such as when the thread has been created, the last run, the number of runs, etc. Note that the information displayed in the Status window is updated at halt.

The VDK Status window (Figure 3-36) is available when a DSP executable is built with VDK support.

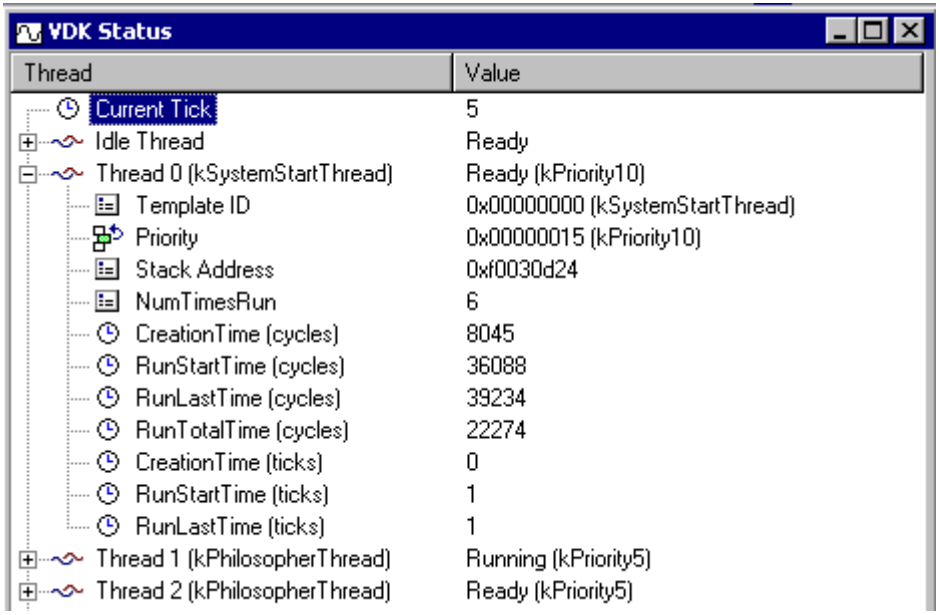

Figure 3-36. VDK Status Window

When a thread is created, it is added to the display. A thread is removed from the display when it is deleted. When you halt the execution of a VDK program, VisualDSP++ reads thread data and displays the thread state and status data in this window.

Initially, thread information appears in a collapsed state in which only the thread name and its current state are displayed. When a thread is in the “Ready “state, its priority is displayed.

Clicking the plus sign next to the thread name expands the view. When thread status information is not accessible (thread out of context), no information is displayed. Possible thread states are listed in Figure 3-33 on page 3-47. For more information about threads, see “Threads” on page 4-2.

General Tips

Even with the data collection features built into the VDK, you may find that debugging thread code is a difficult task. Due to the fact that multiple threads in a system are interacting asynchronously with device drivers, interrupts, and the idle thread, it can become difficult to track down the culprit of an error.

Unfortunately, one of the oldest and easiest debugging methods—inserting breakpoints—can have uncommon side effects in VDK projects. Since multiple threads (either multiple instantiations of the same thread type, or different threads of different thread types) can execute the same function with completely different contexts, the utilization of non-thread-aware breakpoints is diminished. One possible workaround is to insert some ‘thread-specific’ breakpoints:

```
if (VDK_GetThreadID() == <thread_with_bug>)
{
   <some statement>; // Insert breakpoint
}
```